UNIVERSITAT POLITECNICA DE
CATALUNYA

PARALLELISM

Lab 5: Geometric (data) decomposition: heat diffusion equation

Roger Vilaseca Darne and Xavier Martin Ballesteros
PAR4110

7th June 2019, Q2

Contents

1 Introduction

2 Analysis with *Tareador*

In this section we have used the *Tareador* tool to analyse the possible parallelism strategies that we can use in order to parallelise both Jacobi and Gauss-Seidel solvers.

We have mainly focused on the data dependences that appear and how will we protect them in our parallel *OpenMP* code. To explore the dependences, we have used a much finer task decomposition: one task for each iteration of the body of the most innerloop.

2.1 Jacobi Solver

This solver uses an auxiliar matrix **utmp** to write the resulting values of the computation in the most innerloop of the body. For each element in the matrix **u**, it takes the values of the element on its top and bottom and on its left and right, and does some arithmetic operations.

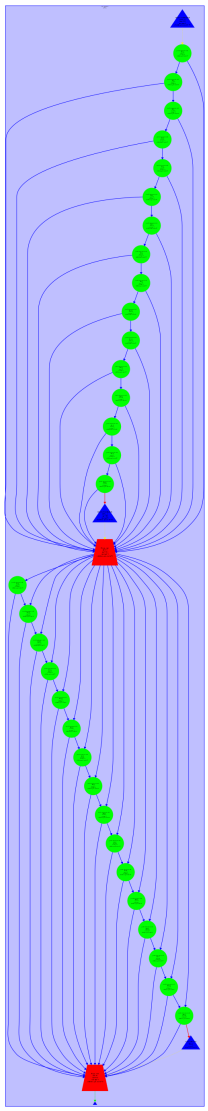
The modified code using the *Tareador* tool is shown below. However, the code can be found in *jacobi-tareador.c* file inside the codes directory.

```
double relax_jacobi (double *u, double *utmp, unsigned sizex,
                     unsigned sizey)
{
    double diff, sum=0.0;

    int howmany=1;
    for (int blockid = 0; blockid < howmany; ++blockid) {
      int i_start = lowerb(blockid, howmany, sizex);
      int i_end = upperb(blockid, howmany, sizex);
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
           tareador_start_task("jacobi_innermost_task");
           utmp[i*sizey+j]= 0.25 * ( u[ i*sizey     + (j-1) ]+  // left
                                     u[ i*sizey     + (j+1) ]+  // right
                                     u[ (i-1)*sizey + j     ]+  // top
                                     u[ (i+1)*sizey + j     ]); // bottom
           diff = utmp[i*sizey+j] - u[i*sizey + j];
           sum += diff * diff;
           tareador_end_task("jacobi_innermost_task");
        }
      }
    }
    return sum;
}
```

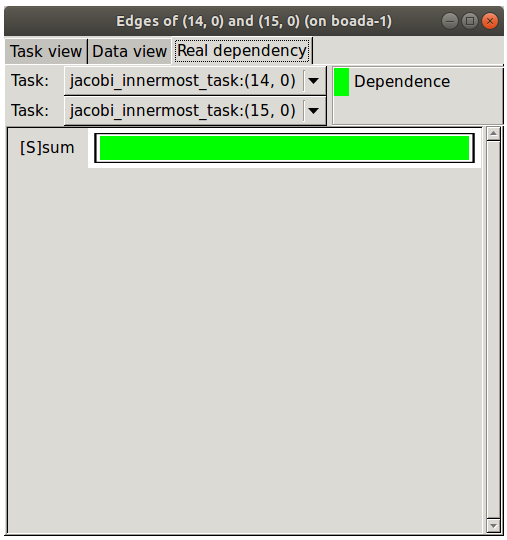
Figure 1: Code for the task decomposition for relax_jacobi function.

With this modified version of the code we can obtain the task decomposition graph (TDG), which can be found in Figure 2a. Moreover we can see that there

exist some kind of data dependencies between tasks. To know which variable is the responsible of these dependences, we have right clicked in an edge between two jacobi_innermost_task nodes (green) >> Dataview >> Edge >> Real dependency. The obtained results are shown in Figure 2b.



(a)



(b)

Figure 2: (a) Task decomposition graph for the Jacobi solver, (b) Real data dependences in the Jacobi solver.

Now, we know that the **sum** variable creates the dependences for the Jacobi solver. Novertheless, we have made use of some *Tareador* calls that temporarily filter the analysis for the variable sum, causing the serialization and obtaining a new task graph (Figure **??**). The modified fragment of the code can be found in *jacobi-tareador-disable-sum.c* file in the codes directory.

```
double relax_jacobi (double *u, double *utmp, unsigned sizex,
                     unsigned sizey)
{
    ...
    for (int blockid = 0; blockid < howmany; ++blockid) {
      ...
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
            tareador_start_task("jacobi_innermost_task");
            ...
            tareador_disable_object(&sum);
            sum += diff * diff;
            tareador_enable_object(&sum);
            tareador_end_task("jacobi_innermost_task");
        }
      }
    }

    ...
}
```

Figure 3: Code for the task decomposition for relax_jacobi function temporarily filtering the analysis of the sum variable.

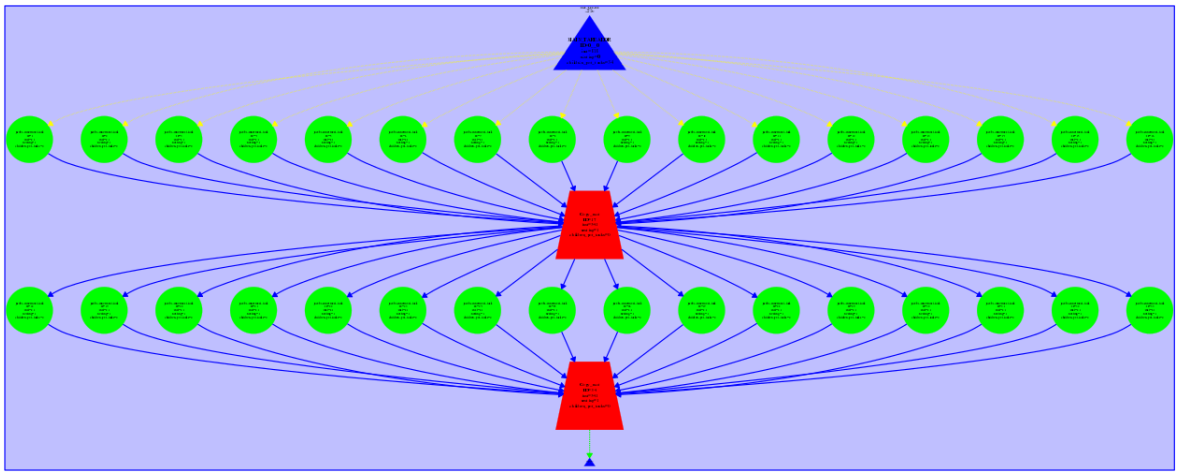


Figure 4: Task decomposition graph of the Jacobi solver temporarily filtering the analysis of the sum variable.

Now, we can see that there is any dependency between tasks of the most inner-loop. Hence, we are increasing the parallelism. We think that the *reduction(+:sum)* clause would be a good option to parallelise the code using *OpenMP* directives.

2.2 Gauss-Seidel Solver

The Gauss-Seidel Solver does no longer use an auxiliar matrix. It writes in the position of the matrix where it reads the values. As in the previous solver, it takes

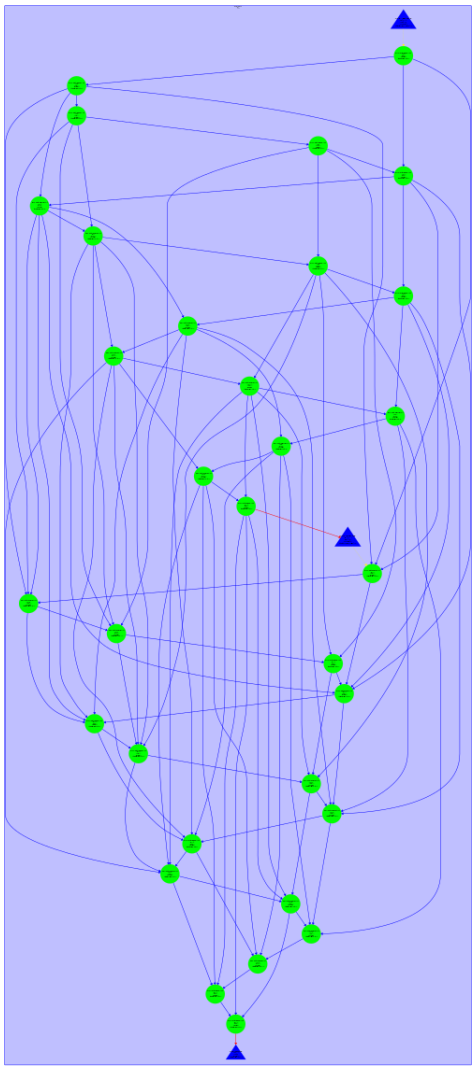
the values of the element on its top and bottom and on its left and right, and does some arithmetic operations.

The modified code can be found in *gauss-seidel-tareador.c* file in the codes directory.

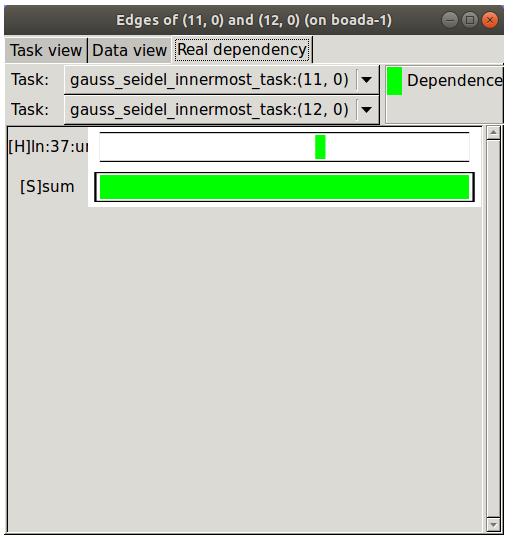
```
double relax_gauss (double *u, unsigned sizex, unsigned sizey)
{
    double unew, diff, sum=0.0;

    int howmany=1;
    for (int blockid = 0; blockid < howmany; ++blockid) {
      int i_start = lowerb(blockid, howmany, sizex);
      int i_end = upperb(blockid, howmany, sizex);
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
            tareador_start_task("gauss_seidel_innermost_task");
            unew= 0.25 * ( u[ i*sizey   + (j-1) ]+  // left
                           u[ i*sizey   + (j+1) ]+  // right
                           u[ (i-1)*sizey  + j     ]+  // top
                           u[ (i+1)*sizey  + j     ]); // bottom
            diff = unew - u[i*sizey+ j];

            sum += diff * diff;

            u[i*sizey+j]=unew;
            tareador_end_task("gauss_seidel_innermost_task");
        }
      }
    }

    return sum;
}
```

Figure 5: Code for the task decomposition for relax_gauss function.

The task decomposition graph is shown in Figure 6a. We can observe that it has also some dependencies. Using the same procedure than in the previous section, we got the Real dependencies for this new solver (Figure 6b).



(a)



(b)

Figure 6: (a) Task decomposition graph for the Gauss-Seidel solver, (b) Real data dependences in the Gauss-Seidel solver.

We can see that this solver has two real dependencies: variable **sum** and some **positions of the matrix**. In this section, we will only show the TDG when disabling only the sum variable. However, the TDG of both variables disables can be found in the Annex section 3.1.

This new version of the code can be found in *gauss-seidel-disable-sum.c*, in the codes directory.

```
double relax_gauss (double *u, unsigned sizex, unsigned sizey)
{
    ...
    for (int blockid = 0; blockid < howmany; ++blockid) {
      ...
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
            tareador_start_task("gauss_seidel_innermost_task");
            ...
            tareador_disable_object(&sum);
            sum += diff * diff;
            tareador_enable_object(&sum);
            ...
            tareador_end_task("gauss_seidel_innermost_task");
        }
      }
    }
    ...
}
```

Figure 7: Code for the task decomposition for relax_gauss function temporarily filtering only the analysis of the sum variable.

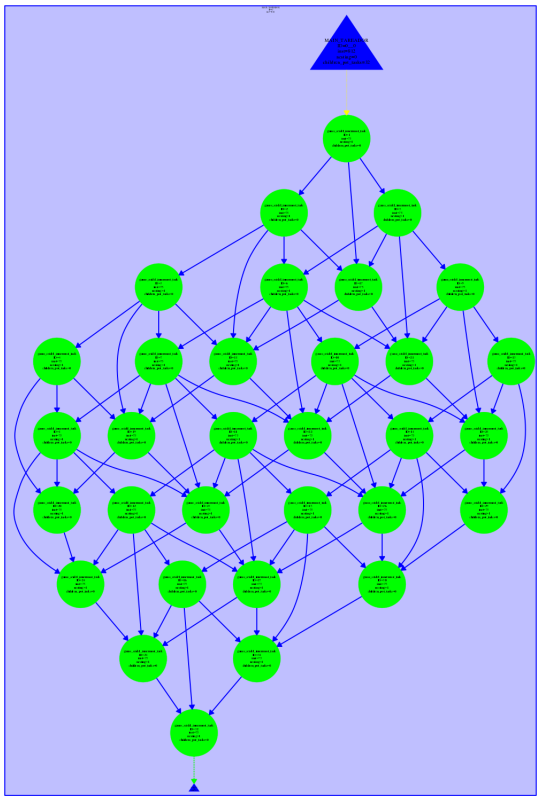


Figure 8: Task decomposition graph of the Gauss-Seidel solver temporarily filtering only the analysis of the sum variable.

Disabling temporarily the variable sum we see that we increase the parallelism.

Again, we think that the *reduction(+:sum)* clause would be good to porallelise the code using *OpenMP* directives. Moreover, we can use the *ordered* clause in order to avoid data races and respect the real dependencies we have seen (apply the doacross technique).

3 Annex

3.1 Task dependency graph when also disabling temporarily some positions of the matrix

In this section we will see the TDG created when disabling all the variables that create some kind of dependency in the Gauss-Seidel solver. The code can be found in *gauss-seidel-disable-sum-and-matrix-positions.c*, inside the codes folder.

```
double relax_gauss (double *u, unsigned sizex, unsigned sizey)
{
    ...
    for (int blockid = 0; blockid < howmany; ++blockid) {
      ...
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
            tareador_start_task("gauss_seidel_innermost_task");

            tareador_disable_object(&u[ i*sizey    + (j-1) ]); // left
            tareador_disable_object(&u[ (i-1)*sizey    + j     ]); //top
            unew= 0.25 * ( u[ i*sizey  + (j-1) ]+  // left
                           u[ i*sizey  + (j+1) ]+  // right
                           u[ (i-1)*sizey  + j     ]+  // top
                           u[ (i+1)*sizey  + j     ]); // bottom
            diff = unew - u[i*sizey+ j];
            tareador_enable_object(&u[ i*sizey + (j-1) ]);
            tareador_enable_object(&u[ (i-1)*sizey + j     ]);

            tareador_disable_object(&sum);
            sum += diff * diff;
            tareador_enable_object(&sum);

            ...
            tareador_end_task("gauss_seidel_innermost_task");
        }
      }
    }
    ...
}
```

Figure 9: Code for the task decomposition for relax_gauss function temporarily filtering the analysis of the sum variable and some positions of the matrix.

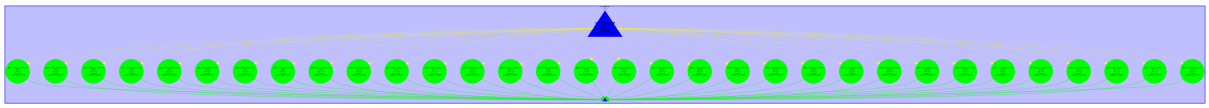


Figure 10: Task decomposition graph of the Gauss-Seidel solver temporarily filtering the analysis of the sum variable and some positions of the matrix.